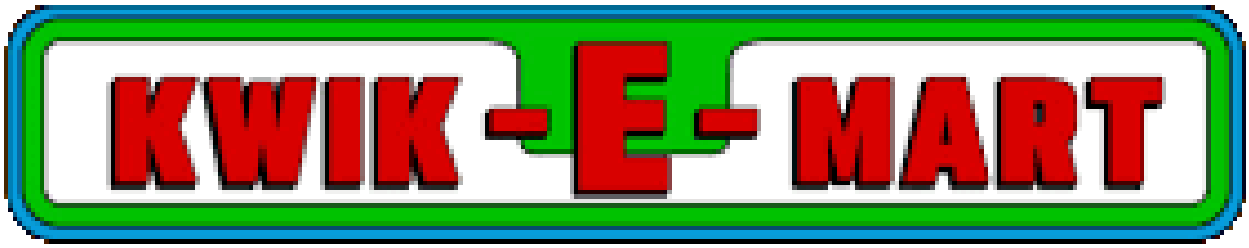
UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

Scuola Politecnica e delle Scienze di Base
Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione

Corso di Laurea Triennale in Informatica
Progetto di Laboratorio di Sistemi Operativi



Cerrone Simone N86004013

Anno Accademico 2023-2024

← Sommario

Analisi dei Requisiti

Nel seguente elenco si riportano i punti chiave espressi in linguaggio naturale delle specifiche di progetto:

- Supermercato con K casse
- Non ci possono essere più di C clienti che fanno acquisti (o che sono in coda alle casse) in ogni istante.
- All'inizio, tutti i clienti entrano contemporaneamente nel supermercato, successivamente, non appena il numero dei clienti scende a $C - E$ $(0 < E < C)$, ne vengono fatti entrare altri E.
- Ogni cliente spende un tempo variabile T all'interno del supermercato per fare acquisti.
- Il cliente si mette in fila in una delle casse … Dopo aver pagato, il cliente esce dal supermercato.
- Ogni cassa attiva ha un cassiere che serve i clienti in ordine FIFO con un certo tempo di servizio.
- I clienti che non hanno acquistato prodotti (P=0), non si mettono in coda alle casse. Ma devono attendere il permesso per uscire.

Dall'analisi di suddette specifiche sono stati estrapolati i requisiti software di cui di seguito descritti:

- Al più C client possono accedere al server contemporaneamente
- La funzionalità di acquisto è limitata a K client alla volta
- Un client, dopo un tempo variabile si mette in coda per accedere alla funzionalità di acquisto
- La funzionalità di acquisto dura $(k * p + c)$ dove k è una costante, p il numero di prodotti del client e c il tempo di servizio del cassiere.
- Se $p = 0$ il client salta la funzionalità di acquisto ed attende un messaggio dal server per uscire.

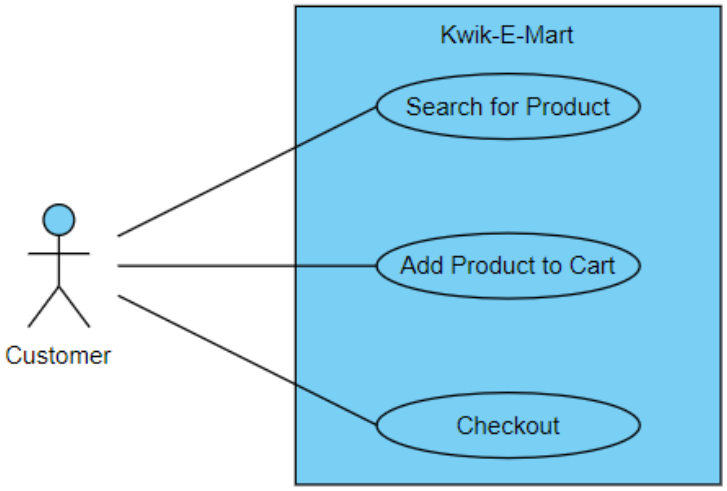


Figura 1:Use Case generale del Sistema

Architettura del Sistema

L'architettura del sistema è stata decomposta in due sottosistemi principali, il Server ed il Client che possiamo rappresentare come di seguito.

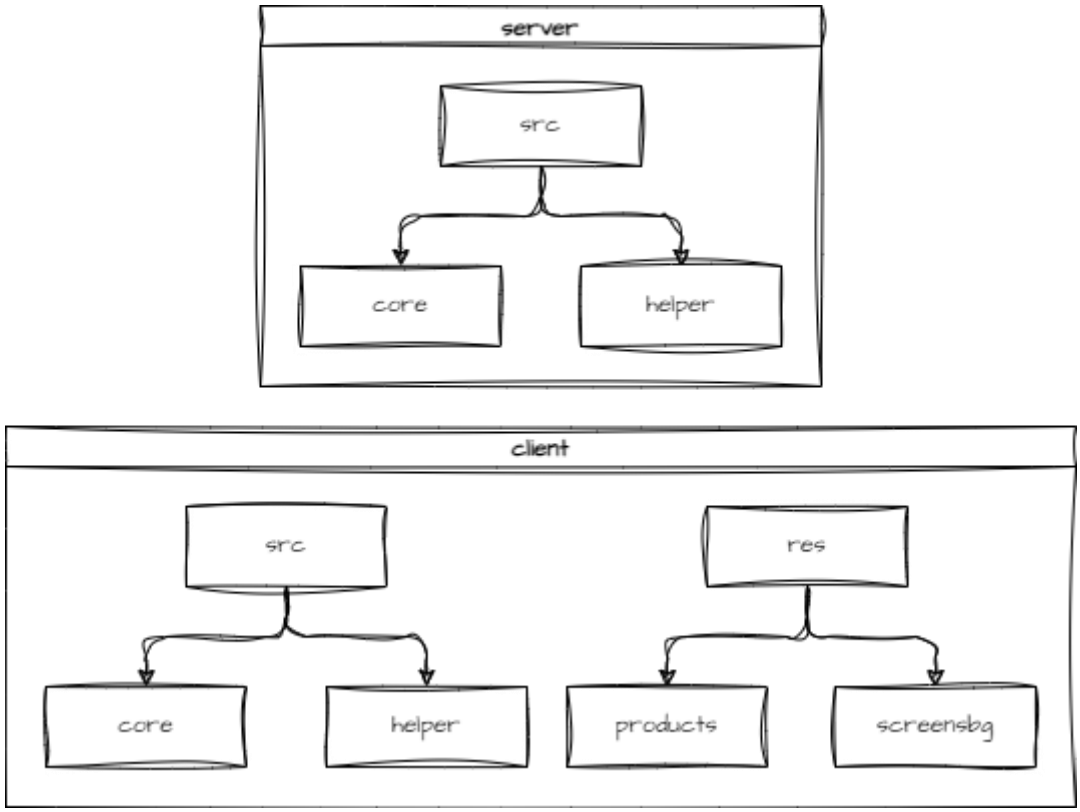


Figura 2: Rappresentazione dei due principali sottosistemi

È evidente la similitudine strutturale dei due pacchetti, il client si differenzia solo per un gruppo di moduli contenenti risorse grafiche. Ma anche se hanno la medesima struttura per la parte contente il codice essi sono molto diversi per contenuto. Nei due successivi paragrafi analizzeremo in dettaglio tali strutture.

Server

- **src**
  contiene i file principali del backend, il *main.c* richiama semplicemente in maniera sequenziale le funzioni scritte in *server.c* anch'esso nella medesima cartella.
  Quest'ultimo contiene la logica del server, implementata sfruttando i seguenti moduli.
  - **core**
    Contiene i componenti essenziali per la gestione del server. In particolare:
    - *psql_api*: gestisce la comunicazione con il database
    - *config*: contiene le definizioni utili alla connessione delle socket
    - *msg_handler:* gestisce le funzioni di comunicazione con le socket
  - **helper**
    Contiene i componenti necessarie al sistema
    - *logging*: contiene funzioni per la stampa di log
    - *signal_handler*: gestione dei segnali di terminazione
    - *ts_queue*: gestione di una coda thread-safe

Client

- **res**
  contiene le risorse grafiche utilizzate per l'interfaccia grafica
  - **products**
    contiene le immagini dei prodotti registrati
  - **screenbg**
    contiene gli sfondi dei vari screen
- **src**
  contiene i file principali del frontend, il *main.py* gestisce la GUI sfruttando la definizione dei widget in *kiwk_e_mart.kv* ed anche la classe contenuta in *client.py,* quest'ultima implementa anche un client automatico eseguito in CLI.
  - **core**
    Contiene un unico file che rappresenta la grafica dei prodotti e la logica base dietro il loro acquisto
  - **helper**
    - *parser*: contiene funzioni di parsing dei prodotti e definizione delle risorse
    - *payload***:** contiene le funzioni wrapper utilizzate nella comunicazione delle socket descritte in *client.py*

Protocollo di Comunicazione

Il protocollo utilizzato per la trasmissione dei pacchetti tra client e server struttura lo scambio dei messaggi come segue.

La comunicazione inizia con l'invio di un messaggio e termina con la ricezione del segnale di ack. Alla ricezione del segnale di nak o dopo un timeout di 7 secondi l'invio del messaggio viene ripetuto.

Il messaggio è una composizione di caratteri ASCII e segue il seguente formato:

SOH	CODE	ARGUMENT	CHECKSUM	CR
0	1	2	$n-2$ $n-1$	n

dove n è la dimensione in byte del messaggio

- **SOH**: carattere ASCII "Start of Heading" che rappresenta l'inizio del messaggio (1 byte)
- **CODE**: Singolo byte che identifica il comando (vedi Tabella 1)
- **ARGUMENT**: Stringa degli argomenti di lunghezza variabile composta da caratteri ASCII stampabili (opzionale)
- **CHECKSUM**: Un byte rappresentato dagli 8 bit meno significativi della somma dei byte che seguono il carattere SOH (incluso)
- **CR**: caratteri ASCII "Carriage Return" che rappresenta la fine del messaggio (1 byte)

Di seguito si descrivono i caratteri di controllo ASCII utilizzati.

DEC	BIN	Simbolo	Descrizione
1	00000001	SOH	Start of Heading
6	00000110	ACK	Acknowledgement
7	00000111	BEL	Bell, Alert
13	00001101	CR	Carriage Return
14	00001110	SO	Shift Out
15	00001111	SI	Shift In
21	00010101	NAK	Negative Acknowledgement

Tabella 1: Caratteri non stampabili della tabella ASCII utilizzati nel protocollo

Prima di descrivere i caratteri non ancora esplicitati poniamo l'attenzione sul fatto che il numero di messaggi che il server scambia con il client è molto limitato; di conseguenza, si è deciso di suddividere lo scambio di messaggi in due fasi: una fase di *inizializzazione*, dove il server invia un comando di ACK se si può avviare la comunicazione o di NAK nel caso in cui si è raggiunto il numero massimo; ed una fase di *comunicazione* dove sarà il client ad eseguire delle richieste, mentre il server risponderà.

Di seguito vengono descritti i comandi delle *request* (con gli argomenti tra parentesi) e le corrispettive *response* del server:

- **BEL** (no args): il client richiede l'accesso al supermercato
    - il server risponde con l'eco del comando e con argomenti il nome e prezzo dei prodotti disponibili. Il separatore di campi sarà '$' mentre quello di componenti '€'
- **SI** (numero prodotti acquistati e prezzo totale con '$' come separatore): il client chiede di mettersi in coda alla cassa
    - Il server dopo un tempo $c * p + k$ (con p numero di prodotti) risponde con **SO**

Dettagli tecnici

In questo capitolo presenteremo alcune delle scelte implementative effettuate in fase di progettazione e implementazione di questo progetto.

Server

Il server è stato sviluppato completamente in C grazie all'ausilio di Makefile e Valgrind per la compilazione ed il debugging. Per la persistenza dei dati è stato scelto il database Postgresql per la sua interfaccia di programmazione di applicazioni C *libpq* (oltre al fatto che è il più avanzato database opensource al mondo).

Il database realizzato è per comodità molto semplice, infatti si presenta come due tabelle indipendenti tra loro e con pochi campi in ognuna, come è evidente nella seguente rappresentazione:

Figura 3: Schema delle entità del database

Tale database è distribuito come container Docker, la configurazione dell'immagine è descritta in un unico file YAML, di seguito riportato, così da semplificare la creazione e l'avvio dei servizi tramite docker-compose.

Client

Per un rapido sviluppo del client si è scelto di utilizzare il linguaggio di programmazione Python. In particolare, l'interfaccia grafica è stata realizzata con il framework Kivy che permette una facile realizzazione di app GUI python multipiattaforma.

Utilizzo e Sorgenti

È possibile visionare il sorgente nella seguente repository github: **Kwik-E-Mart**

Per l'utilizzo del sistema, nonché le dipendenze utilizzate si rimanda alla lettura del file README.dm visualizzabile nello stesso git.